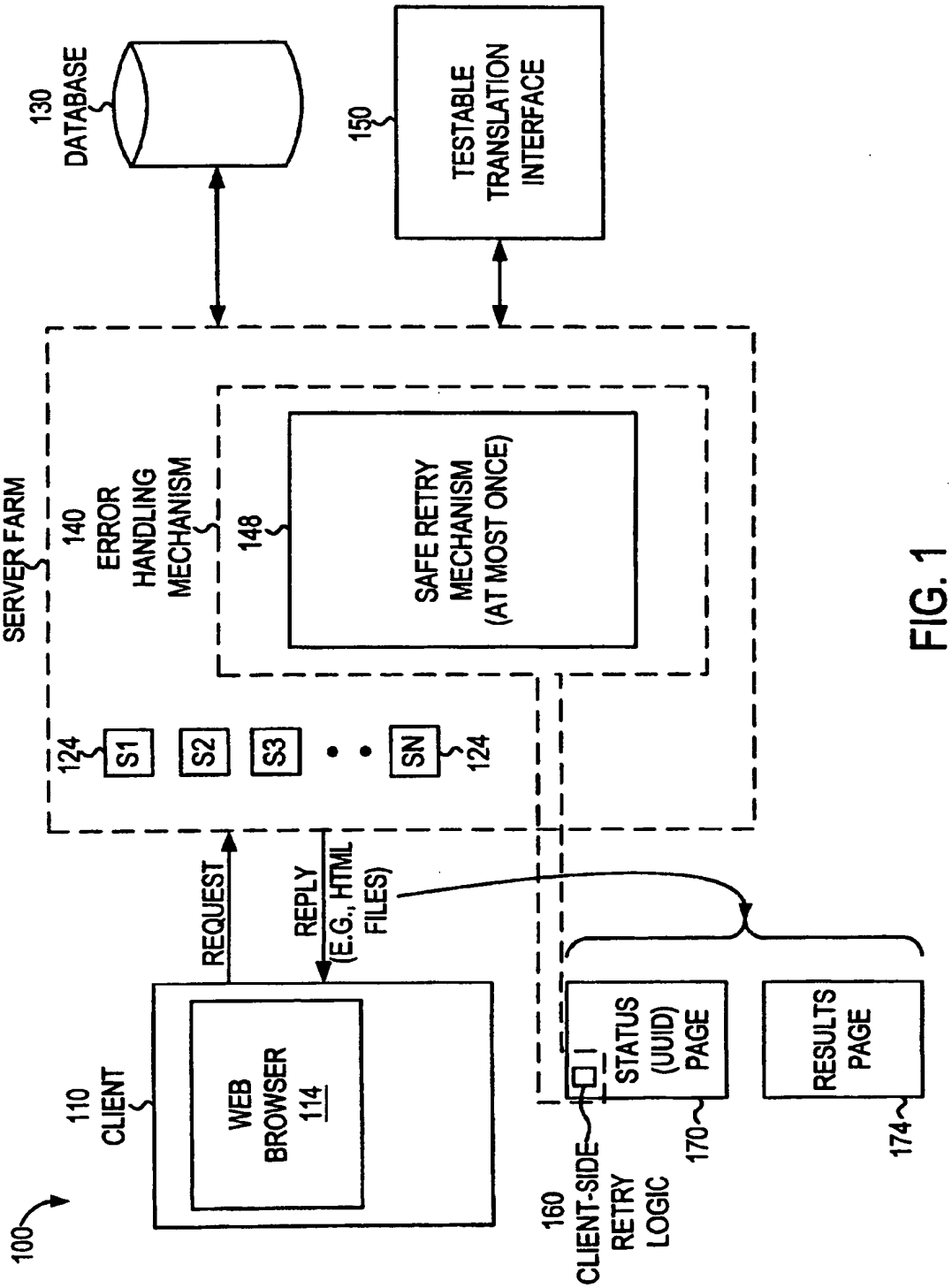

FIG. 1

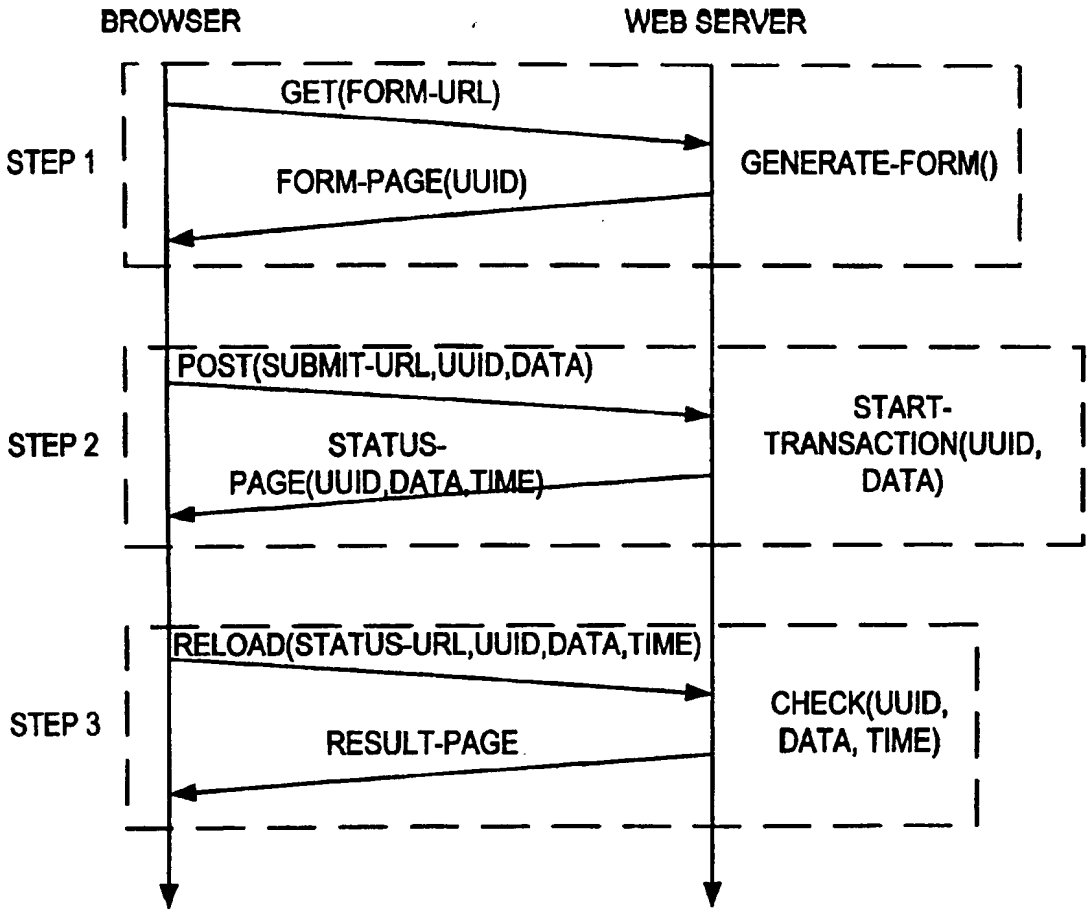
BROWSER
WEB SERVER
STEP 1
GET(FORM-URL)
FORM-PAGE(UUID)
GENERATE-FORM()
STEP 2
POST(SUBMIT-URL,UUID,DATA)
STATUS-PAGE(UUID,DATA,TIME)
START-TRANSACTION(UUID, DATA)
STEP 3
RELOAD(STATUS-URL,UUID,DATA,TIME)
RESULT-PAGE
CHECK(UUID, DATA, TIME)

FIG. 2

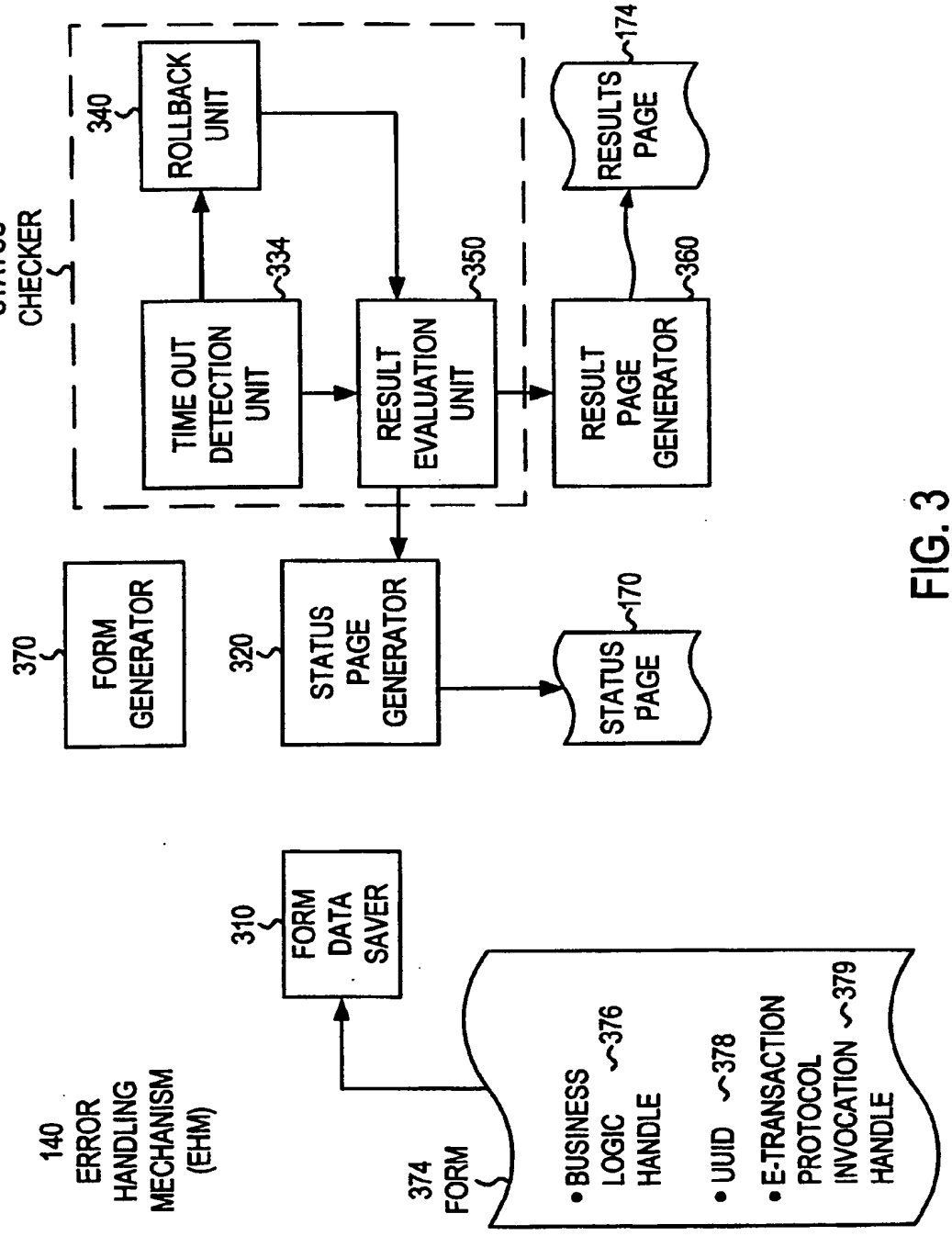
140
ERROR HANDLING MECHANISM (EHM)
310
FORM DATA SAVER
374
FORM
• BUSINESS LOGIC HANDLE 376
• UUID 378
• E-TRANSACTION PROTOCOL INVOCATION HANDLE 379
330
STATUS CHECKER
340
ROLLBACK UNIT
334
TIME OUT DETECTION UNIT
350
RESULT EVALUATION UNIT
360
RESULT PAGE GENERATOR
174
RESULTS PAGE
370
FORM GENERATOR
320
STATUS PAGE GENERATOR
170
STATUS PAGE

FIG. 3

```
html generate-form() {
  uuid := new UUID();
  return form-html(uuid);
}

void biz-logic(uuid,data) {
  start(uuid);
  // Execute SQL with data as argument.
  // Store result in the variable res
  testable::commit(res,uuid);
}

html start-transaction(uuid,data) {
  // Spawn new thread to execute biz-logic
  // with uuid and data as arguments
  time := current-time();
  return stat-page-html(uuid,data,time);
}

html check(uuid,data,time) {
  if current-time() - time > timeout then
    testable::rollback(uuid);
    res := testable::get-outcome(uuid);
    if res == nil then
      return start-transaction(uuid,data);
    else
      return result-page-html(res);
  else
    res := testable::get-outcome(uuid);
    if res != nil then
      return result-page-html(res);
    else
      return stat-page-html(uuid,data,time);
 }
```

FIG. 4

```
void FormServlet(HttpServletRequest req,
            HttpServletResponse resp) {
  request.getSession(true);
  // Normal Form Creation processing with Form handling servlet name stored
  // in a hidden field, and action set to the Start servlet}
void BizLogic(HttpServletRequest req,
          HttpServletResponse resp) {
  WeTDriver.getConnection(``JDBC-URL");
  // Execute JDBC commands and other logic
  // Output result normally; Do NOT commit any JDBC updates}
void Start(HttpServletRequest req,
        HttpServletResponse resp) {
  WorkQueue.queue(req.clone(),
              resp.clone());
  SendStatPage(req, resp);}
void Check(HttpServletRequest req,
        HttpServletResponse resp) {
  result = getOutcome(req.getSession());
  if (result != null) \{
    result.send(resp);
    return;}
  job = getJobFromQueue(req.getSession());
  if (currentTime() - job.time > timeout) {
    job.abort();
    WorkQueue.queue(job.req, job.resp);
  } else {SendStatPage(job.req, job.resp);}}
void WorkerThread() {
  while(true) {
    job = getJobFromQueue();
    servlet = job.targetServlet;
    try {
      // Begin Transaction
      servlet.service(job.req, job.resp);
      storeOutcome(job.req, job.resp);
      removeJobFromQueue(job);
      // Commit Transaction
    } catch (Exception e) {
      // Abort transaction}}}
```

FIG. 5